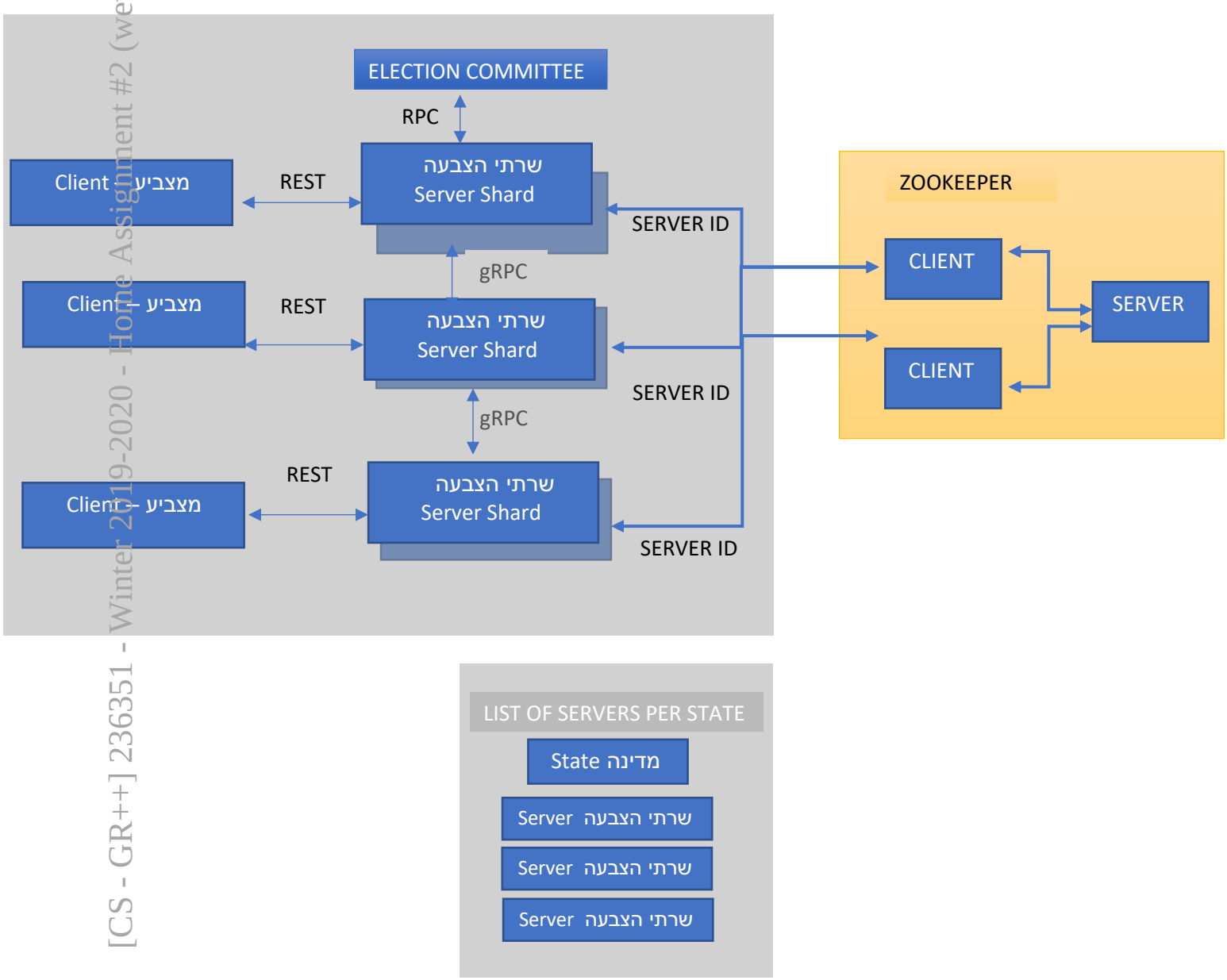
מטרת המסמך

במסמך זה נתאר בצורה טכנית את המערכת שבנינו ל-Voting Election.

תחילה, נגדיר את הישויות והאובייקטים הרלוונטיים להמשך קריאת המסמך. נמשיך בהגדרת מוסכמות וקווי התנהגות הנוגעים לאופן מילוי הבלוקים ובדיקת נכונותם. לאחר מכן נתאר מספר ממשקים וביניהם:

1. ממשק לקוח – שרת הצבעה.
2. ממשק בין שרתי ההצבעה השונים.
3. ממשק בין שרתי ההצבעה השונים ל-zookeeper אשר מהווה שירות חיצוני למערכת ההצבעה.
4. ממשק בין השרתים ל-election committee בשביל לקבל הודעות סטטוס אודות נתוני הצבעות.



מבוא ומושגים בסיסיים

להלן ההגדרות של מספר מושגים בסיסיים שבהם נשתמש בהמשך:

- **טרנזקציה (transaction)** –הצבעה של voter_id מסוים כלפי nominee מסוים.
- **Zookeeper** – תשתית סנכרון נתונים מבוזרת שמיועדת לשיתוף הגדרות ונתונים בין שרתים שונים ומאפשרת הרשמה להתראות על שינויי הגדרות במערכת (למשל על נפילת שרת).
- **שרת הצבעה -** שרת הצבעה מנהל את המידע על הצבעות המצביעים באמצעות החזקת מיפוי מה-ID של כל מצביע לstate שלו. כל קבוצת שרתים shard מוגדרת כמדינה, אשר מנהלת בצורה אוטונומית את מספר המצביעים אליה עבור מועמד מסוים. השרת מנהל את ההצבעות באמצעות. Map<Integar,Integar> finalVotes אשר שומר עבור הshard הספיציפי.
- **Election Committee** –שרת אשר יקבל את תוצאות המנצח מכל שרד ויחזיר תשובה מי המנצח של הבחירות.

ולידיות של טרנסאקציה (הצבעה)

בעת קבלה של טרנזקציה, נרצה לוודא כי היא עומדת בתנאים שהגדרנו, בכדי להימנע מיצירה של בלוק שאינו ולידי. לצורך התרגיל, הנחנו כי כל טרנזקציה חייבת להכיל את השדות המתאימים, ואותם בלבד (הid של המצביע, המדינה אליה שייך ולמי הצביע). במידה ויש בעיה, תיזרק חריגה, שבתורה תהפוך לגוף ההודעה בתגובת ממשק ה-REST, ותיידע את הלקוח מה הבעיה.

ממשקים

1. ממשק לקוח – שרת הצבעה:

התקשורת בין המצביעים לבין השרתים שאיתם הם מדברים נעשית באמצעות REST API הממומש באמצעות Spring. אנחנו מאפשרים למצביעים לבצע מספר פעולות מרכזיות, המתאימות לפעולות שנתמכות בממשק REST.

rest_uri = http:// IP_ADDRESS: (START_PORT_REST+SERVER_NUMBER)/NOMINEE/VOTER_ID

- PUT: נעזר בפונקציה put ונכניס לה את rest_uri וכך אנו נבצע הצבעה עם ממשק rest.

כל הצבעה במערכת היא מהצורה הנ"ל:

VoteRequst
FromServer
Nominee
VoterID

סוגי messages נוספים אשר יש במערכת:

Status
SUCCESS
FAILURE

ElectionPeriod
START_ELECTION
END_ELECTION

ReportForState
Nominee

StateID
Nominee

2. ממשק בין שרתי ההצבעות :

a. שליחת הצבעה בין שרתים (בין קבוצות שרתים)

לצורך שליחת הבלוקים עצמם בין שרתי ההצבעה השונים, השתמשנו בתשתית ה gRPC. בכדי לאפשר את התקשורת הנדרשת לדרך השליחה שלנו, נתחזק רשת mash (גרף הרשת, יהיה גרף מכוון ומלא). כלומר, בין כל שני שרתים יהיה חיבור לכל כיוון. לצורך השליחה עצמה, השתמשנו ב future stub, במטרה לא לחסום את המשך ריצת השרת בכל שליחה לשרת אחר. בנוסף, בכדי לייעל, רצינו להימנע מחסימה עד שכל השליחות של בלוק מסוים יסתיימו, לכן הוספנו thread חדש (Sender), אשר כל תפקידו לבצע את השליחה. בצורה זו השרת יכול לשלוח מספר בלוקים במקביל ולהקטין את ה- latency של המערכת.

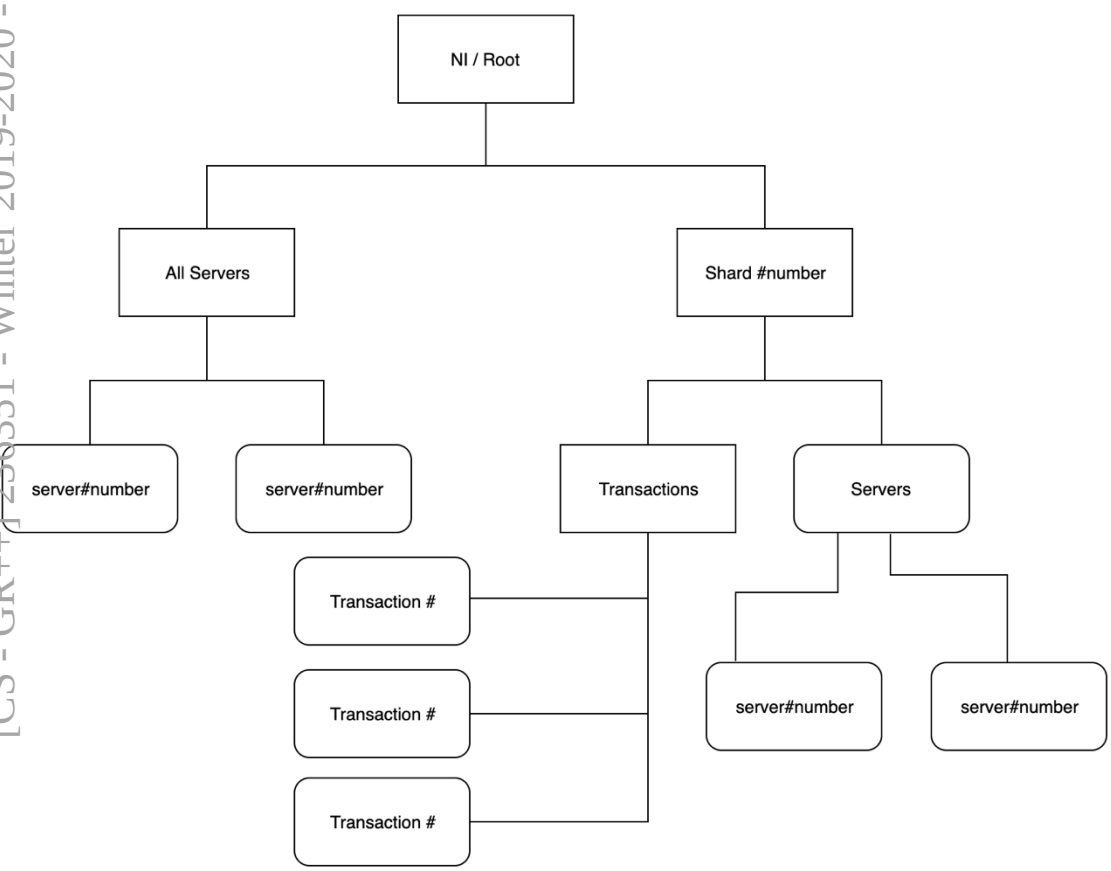
ה Sender בריצתו, משתמש ב future stubs לצורך השליחה בתוכו, וכך שולח לכל השרתים האחרים. התוצאות (future) יישמרו במילון מתאים לפי מזהה השרת המתאים. לאחר מכן, כל עוד המילון הנ"ל לא ריק, יבדוק ה Sender לכל אחד מהערכים בו, האם השליחה הסתיימה או שהשרת המקבל נפל, במידה וכן, יסיר את התוצאה מהרשימה, וימשיך. כל הממשק הנ״ל מתקיים במידה ומצביע ביקש להצביע בשרת שאינו שייך לשרד שאליו המצביע משתייך.

3. ממשק בין שרתי ההצבעות ל-Zookeeper:

Zookeeper היא תשתית סנכרון נתונים מבוזרת שמיועדת לשיתוף הגדרות ונתונים בין שרתים שונים ומאפשרת הרשמה להתראות על שינויי הגדרות במערכת (למשל על נפילת שרת). אנחנו נשתמש בתשתית זו לשתי מטרות:

1. Producer Consumer Queue
2. Membership Management

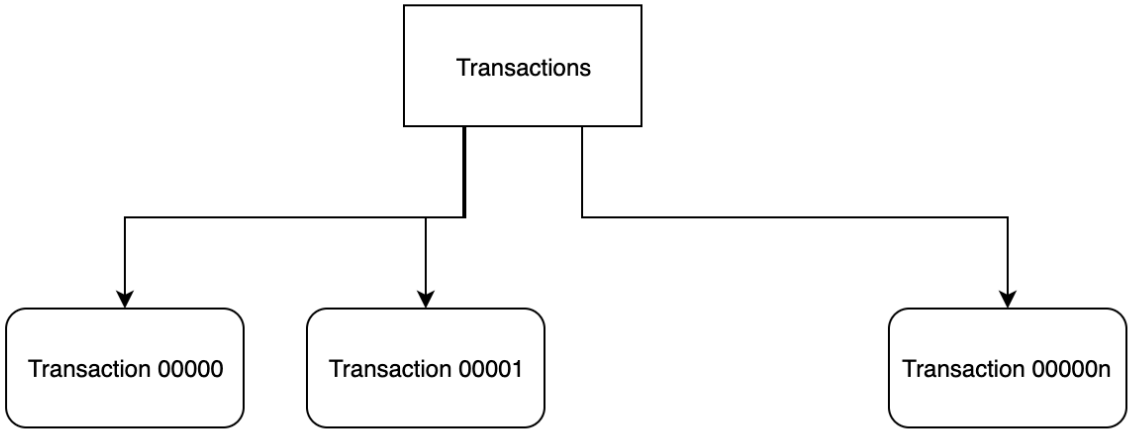
מבט כללי על המערכת:



1. Producer Consumer Queue

בכדי לממש את המנגנון הנ"ל החלטנו להיעזר בתשתיות של שירות ה-zookeeper אשר משתמש במנגנון Sequtial node (זאת ע"י מימוש Paxos מאחוריי הקלעים). לצורך מימוש של queue. Many to many.

התור ממומש ע"י zNode ראשי בשם "Transactions" אשר שם ההעברה הוא מספרי מונוטוני עולה. כל העברה מזוהה בתור מי הצביע, היכן הצביע ומה הצביע. נבחין כי כל zNode בהיררכיה של ZK יכול להחזיק כמות קטנה של מידע ולכן לא יכולנו לאכלס בלוק שאליו מוצמדות מספר רב של טרנזקציות בתוך חוליה בעץ. לכן כל בלוק בzk הינו הצבעה בודדת. כאשר שרת הצבעה רוצה להעביר עליו לייצר העברה חדשה בתור. כאשר תגיע תורה והעברה זו תהיה ההעברה הישנה ביותר, כל השרתים יבצעו עליה את פרוטוקול 2 phase commit כלומר רק לאחר שווידא כי ההעברה התקבלה אצל כלל השרתים החיים, יכול הלידר למחוק את הטרנסאקציה מהתור. נוצר zNode חדש מסוג PERSISTENT_SEQUENTIAL. בנוסף לתכונת sequential , דגל PERSISTENT יאפשר לשרתים השונים לבצע העברות שמגיעות משרת שייתכן כי נפל, כל עוד הספיק לפרסם את הבלוק ל-ZK. כלומר- כל העברות ששרת כלשהו הספיק לשלוח טרם נפילתו יישארו "חתומים" בשרשרת הבלוקים ולא ייגנזו עם הנפילה.



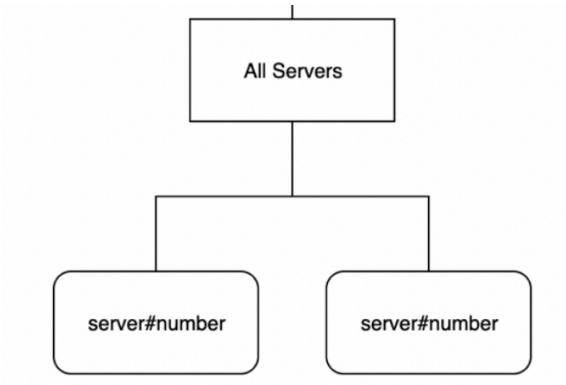
נשים לב שהודות למזהים הייחודים של zNodes תחת Transactions מיונם מניב רצף חד משמעי של סדרי ההצבעות ורצף זה מהווה בעצם את סדר ההצבעות, כלומר סוג של time-stamp. כל שרת שברצונו לחשב את המנצח בשרד שלו, מסדר את ההצבעות שנשלחו אליו קודם לכן מיתר השרתים לפי הסדר שמשרה מיון.

2. Membership Management

שימוש נוסף שעשינו ב-ZK הינו למעקב, סנכרון ואתחול של המערכת.

מעקב אחרי קבוצת השרתים החיים - את התמיכה מימשנו באמצעות תיקיית "servers" אשר כל ה-zNodes שיושבים תחתיה מייצגים שרתי הצבעות פעילים במערכת. בעת הפעלת שרת הצבעות אחת מפעולות האתחול שלו היא הרשמה למערכת אשר מייצרת בעבורו zNode מסוג EPHEMERAL_SEQUENTIAL. תכונת ה-EPHEMERALמבטיחה כי ברגע שהשרת ייפול החוליה תיעלם גם היא מהתיקייה וכך השרתים השונים יוכלו לבצע מעקב אחר השרתים החיים. הערך של כל zNode הוא ה-ID של השרת ואנו מניחים כי אין כפילות ב-ID.

כך למשל תיראה התיקייה עבור מערכת שכללה מספר שרתי הצבעות שונים.



את ההבטחה לאתחול של המערכת עם כלל השרתים, מימשנו באמצעות barrier. Barrier שמבוסס על הzk. תפקיד הbarrier לוודא את הצטרפות כלל השרתים וגם שכל השרתים הספיקו לשתף מידע בינהם אודות מספר הפורטים, ורק אז מותר להתחיל לקבל בקשות start election שונות.

סנכרון - מנגנון זה מומש כ-watcher בכדי לאפשר עדכונים על נפילה של שרתים למערכת. העדכון המיידי על שינויים מסוג זה מאפשר לכל שרת להחזיק סט עדכני של שרתים שהוא מכיר ויכול לתקשר איתם. סט זה רלוונטי מבחינתו הן לצרכי נכונות והן לצרכי יעילות.

ממשק servers – committee

הcommittee יכול לתקשר עם הסרברים השונים באמצעות grpc, ולקבל נתוני הצבעות עבור השרדים אשר הם שותפים בהם. באמצעות ממשק זה נדע מי המנצח של הבחירות.

הcommittee שולח בקשה לשרת כלשהו בודד מכל קבוצה, מקבל תשובה מי ניצח עבור הקבוצה הרלוונטית, ומחזיר מי המנצח המקסימלי. במידה אם השרת שהוא ניסה לשלוח אליו לא מתפקד, ינסה הcommittee לשלוח לשרת אחר.

סיכום

יצרנו מערכת לניהול הצבעות וביצוע פעולות בסיסיות בין שרתים. כל שרת של הצבעות, ישמור אצלו (בנוסף לנתונים הבסיסים) את מצב ההצבעות, ובהכרח באותו סדר כמו אצל השרתים האחרים. ביצוע ההצבעות עצמן, ייתבצע גם הוא בסדר זהה (לא בהכרח סדר השליחה מהלקוח) אצל כולם,.

לצורך המימוש, השתמשנו ככל הניתן בלוגיקה לא חוסמת. כלומר, השתמשנו ב threads נפרדים לשליחה של כל בלוק, ובתוך כל thread כזה, השתמשנו ב Future Stub, שאינו חוסם. כך קיבלנו מערכת, שבשגרה, תעבוד באופן מקבילי (ככל הניתן, בהתאם למכונה המריצה), ובכך תייעל את אופן התקשורת ותפחית משמעותית את הזמן בו השירות אינו זמין ללקוח.

כיווני המשך אפשריים

לייצר מנגנון של batching – נסביר

בעת בחירת ההצבעות אשר ייכנסו לבלוק הצבעה מסוים לשליחה נכנס לשיקול כמות ההצבעות אותה נרצה להכניס. מצד אחד, נרצה למלא בלוק עם כמות גדולה של הצבעות בכדי ל"הוזיל" השליחה היקרה ל-zookeeper כלומר, הצמדת מספר הצבעות מפחית את התקורה המשוערכת פר הצבעה ובכך מגדיל את ה-throughput של המערכת. מנגד, כמות גדולה מדי של הצבעות תעכב את שליחת ההצבעות ברשת כיוון שבמקום לשלוח הצבעה ברגע שבו היא מגיעה למערכת אנחנו מחכים להצבעות אחרות שיתווספו אליה. בכדי למצוא איזון מוצלח בין הצדדים ניתן לממש מנגנון שנקרא adaptive batching, או ליתר דיוק timed-adaptive batching כפי שהוצג במאמר "Adaptive Batching for Replicated Servers" שפורסם על ידי רועי פרידמן וארז חדד בשנת 2006. רצוי לומר שעדיין נצטרך לשמור על סדר בין הצבעות שונות בשביל לוודא את אמיתות ההצבעה האחרונה עבור מצביע בודד.

READ ME – עבור ממשק הפונקציות:

הרצה משורת הפקודה בterminal –

Python start_servers.py

נחכה עד ששרת הzookeeper יעלה.

לאחר ששרת הzookeeper עלה, נלחץ enter ונגדיר את מספר המדינות ומספר השרתים שיש לכל מדינה.

נחכה ששרתי המדינות וממשק הspring יסיימו לעלות ויהיו בהאזנה.

כעת אנו יכולים להעיזר במתודות הממשק אשר הגדרנו –

“vote(server_number, voter_id, nomiee)” – **vote**	לבצע הצבעה בשרת מספר server_number מvoter_id למועמד nominee.
Test **test**	לבדוק מספר הצבעות רנדומליות רצופות.
Close_server(number) - **kill**	לסגור את שרת מספר number.
exit_ – **exit**	סגור את כל התהליכים אשר יצרנו.
View_report – **report**	מדווח את מספר ההצבעות כרגע שנספרו במערכת ואומר מי המנצח.
view_report_shard- **sreport**	מדווח עבור מדינה(shard) ספיציפי את המנצח.
Election_end - **start**	מסיים תקופת הצבעה.
Election_start - **end**	מתחיל תקופת הצבעה

דגשים:

קובץ nominees.txt צריך להיות באותה התיקייה של קובץ הstart_server.py, כל שורה לאחר ה# צריכה להיות ערך מספרי, שמציין מועמד. בכל שורה יש מספר בודד, ואין רווחים בין השורות.